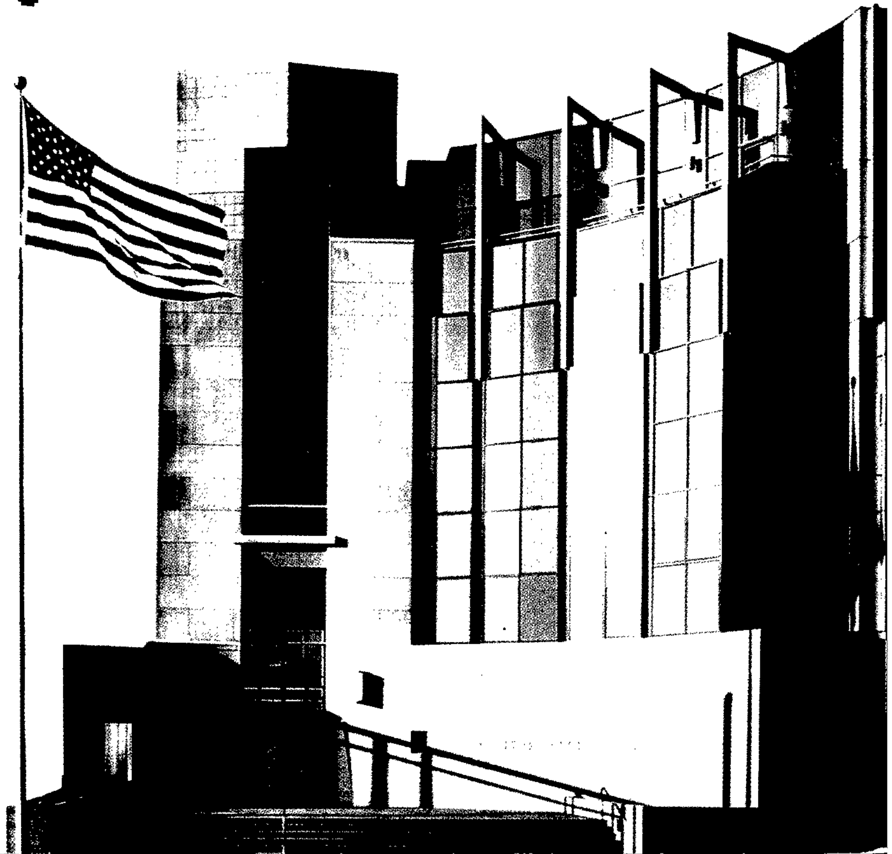
Carnegie Mellon
Software Engineering Institute

Agora: A Search Engine for Software Components

Robert C. Seacord
Scott A. Hissam
Kurt C. Wallnau

August 1998



TECHNICAL REPORT
CMU/SEI-98-TR-011
ESC-TR-98-011

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Agora: A Search Engine for Software Components

CMU/SEI-98-TR-011
ESC-TR-98-011

Robert C. Seacord
Scott A. Hissam
Kurt C. Wallnau

August 1998

Dynamic Systems

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

[illegible]

Mario Moya, Maj, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

List of Tables

Acknowledgements

Neil Christopher at the National Institute of Standards and Technology Manufacturing Engineering Lab (NIST/MEL) sponsored this work. Special thanks to John Foreman for supporting this work and SEI reviewers Jeromy Carriere and Suzanne Couturiaux.

Abstract

Agora is a software prototype being developed by the Commercial Off-the-Shelf (COTS)-Based Systems Initiative at the Software Engineering Institute (SEI). The object of this work is to create an automatically generated and indexed worldwide database of software products classified by component model. Agora combines introspection with Web search engines to reduce the costs of bringing software components to, and finding components in, the software marketplace. This report describes Agora's role in an emerging component industry and the features and capabilities provided by Agora. The implementations of a JavaBeans agent and a Common Object Request Broker Architecture (CORBA) agent are also described. These agents are used to gather components of their respective types.

1 An Emerging Component Industry

1.1 Background

Software developers welcome the emergence of a robust marketplace of software components, as a highly competitive marketplace works to the advantage of both producers and consumers of software components. However, two essential requirements for a component marketplace have been slow to emerge: standard, interchangeable parts, and the consumers' ability to find the right parts for the job at hand. Fortunately, recent advances in component and Web technology are, at last, providing the means for satisfying these requirements. Component technology, such as JavaBeans and ActiveX, provides a basis for interchangeable parts, while the Web provides a means for consumers to locate available components.

1.2 Agora

Agora is a prototype being developed by the Software Engineering Institute at Carnegie Mellon University. The object of this work is to create an automatically generated, indexed, worldwide database of software products classified by component type (e.g., JavaBean or ActiveX control). Agora combines introspection with Web search engines to reduce the costs of bringing software components to, and finding components in, the software marketplace.

Introspection is a term primarily associated with JavaBeans that describes the ability of JavaBeans to provide information about their own interfaces. Common Object Request Broker Architecture (CORBA) provides a similar capability of divulging information about interfaces, although these data are maintained external to the CORBA server in an implementation repository.

Until recently, surfing was a typical approach for finding information on the Web. Search engines, such as Webcrawler, Lycos [Maudlin 97], AltaVista, and InfoSeek, enable users to search for and locate information published on the Web more effectively. Search engines use indexes that are automatically compiled by computer programs (such as robots and spiders) and that go out over the Internet to discover and collect Internet resources. Searchers can connect to a search engine site and enter keywords to query the index. Web pages and other Internet resources that satisfy the query are then identified and listed [Webster 96].

The combination of introspection with component search is a necessary but insufficient element of an online component marketplace. Elements required by a component marketplace that are not addressed by this work include security, electronic commerce, and quality assurance. The National Institute of Standards and Technology (NIST) Advanced Technology Program (ATP) Focused Program on electronic commerce [Hurwitz 98] is already addressing some of these concerns. This work is taking place in the context of NIST programs in component-based software [Nowak 97] and the NIST Laboratory program (Manufacturing Engineering Laboratory, Manufacturing Systems Integration Division) interest in software components for the manufacturing domain [SIMA 97, NIST 98].

Even modest steps towards integrating component technology and Web search can have an impact on the emergence of an online component marketplace by

- providing developers with a worldwide distribution channel for software components
- providing consumers with a flexible search capability over a large base of available components
- providing a basis for the emergence of value-added component qualification services, within and across specific business sectors

2 Features and Capabilities

Agora supports two basic processes: the location and indexing of components and the search and retrieval of a component. The location and indexing of components is primarily an automated background task, while a human typically performs search and retrieval. There are exceptions in that an interface exists to allow a vendor to add a specific component to the index.

2.1 Location and Indexing

Agora uses a variety of agents for locating and indexing component information. Currently a JavaBeans agent and a CORBA agent have been developed, each able to locate and index components of their respective type.

Components are introspected during the indexing phase to discover their interfaces. Introspection of JavaBeans is accomplished using the mechanism provided by JavaBeans Introspector class. In CORBA, interface information is maintained separately in an interface repository. As a result, this information may not be available because it is not in the repository, the repository is not running, or the information cannot be located. In each of these cases, the interface information cannot be successfully retrieved and indexed.

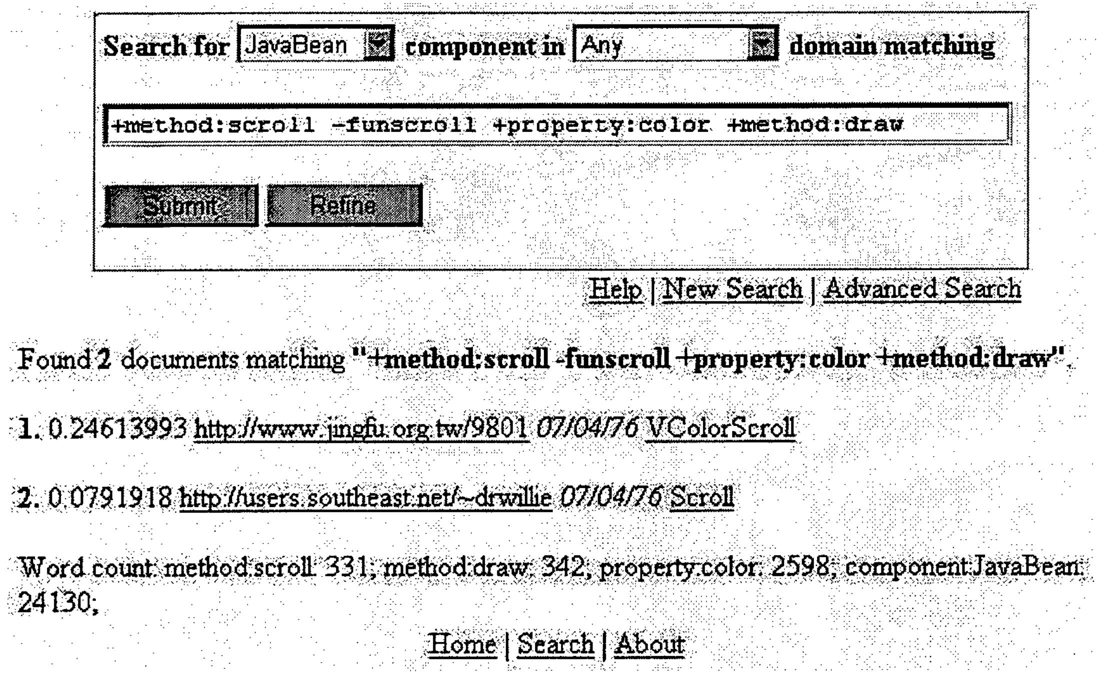
Once a component has been identified, the interface information is decomposed into a set of tokens. A document is created in the index that includes these tokens. Unlike a text document, component interface information can be differentiated into different fields. Examples of fields may be methods, attributes, or events. This information is also maintained for each component to enable specialized searches to be performed. The component name and type are also preserved as fields to enable searches by name and component type. Meta-information about each component is also maintained with the document, including the Uniform Resource Locator (URL) for each component. Maintaining the component URL allows detailed interface information to be re-collected during the search and retrieval process and allows the user to examine, in the case of hypertext transfer protocol (HTTP)-based URLs, the Web page containing the component.

2.2 Search and Retrieval

Search and retrieval in Agora is a two-step process. Initially a searcher enters query terms and optionally specifies the type of component. These terms and other criteria are searched against the index collected by the search agents. It is also possible to issue a *field* search to find a term in a particular context (for example, to find components of a given name or components that implement a given component model). The result set for the query is sent back to the user for inspection. Each result includes meta-information including the URL of the component. The searcher can then refine or broaden the search criteria based on the number and quality of the matches.

Once the searcher has completed this breadth-wide search to identify candidate components, individual components can be examined in detail. The URL for the component is returned to the Component Introspector for re-introspection. This approach reduces the amount of information that must be maintained in the index for each indexed component, ensures that the component is still available at the specified location (URL), and guarantees that an up-to-date interface description is retrieved.

Figure 1 shows the results from a JavaBeans search. In this case, the search criterion specifies that the JavaBean must contain methods `color` and `draw` and property color but must not contain the term `funscroll`. This search resulted in two documents being found, each with relatively low relevancy ranking. The word count indicates how many occurrences of each term were found in the database.


Search for JavaBean component in Any domain matching
Submit
Refine
Help | New Search | Advanced Search
Home | Search | About

```
+method:scroll -funscroll +property:color +method:draw
```

Found 2 documents matching "+method:scroll -funscroll +property:color +method:draw".

1. 0.24613993 http://www.jingfu.org.tw/9801 07/04/76 VColorScroll

2. 0.0791918 http://users.southeast.net/~drwillie 07/04/76 Scroll

Word count: method:scroll: 331; method:draw: 342; property:color: 2598; component:JavaBean: 24130;

Figure 1: Agora Query Interface

Once a suitable result list has been returned, the searcher can select the link to the component's URL. This normally allows the searcher to see what the component looks like when it is operational and possibly collect additional information about the component. Selecting the component name causes the component's interface description to be displayed.

2.3 Advanced Search Features

Agora supports the basic operators '+' and '-': these operators indicate words or phrases that are required or prohibited in the search results. Agora also allows advanced search capabilities that support the Boolean logic operators AND, OR, NOT, and NEAR, as well as the ability to specify ranking words that are different from the words in the search.

Agora supports a number of special functions that allow the user to narrow the search criteria using specific characteristics of the component. Table 1 shows special functions that operate across all component types.

Keyword	**Function**
component:type	Finds components of this type. Each type corresponds to some agent.
name:name	Finds components with this name. This is equivalent to the applet function in AltaVista.

Table 1: JavaBeans Special Functions for All Component Types

For, example, the user may restrict a query to search for components implemented as JavaBeans by using the component function to specify the component type with the following query:

```
component:JavaBeans
```

In addition to special functions such as component that operate across all component types, there are also component-specific special functions. These functions are specific to each component type, supporting component-specific capabilities and the use of domain-specific terms.

For example, Agora supports the special functions shown in Table 2 for CORBA interfaces. The CORBA special functions for *operation* and *attribute* map to the Java special functions for *method* and *property,* shown in Table 3. Different names are used for these functions to match more precisely to domain-specific nomenclature. There is no mapping between the JavaBeans *event* function and CORBA. Special functions for *exception* and *parameter* in CORBA derive naturally from the capabilities of the interface repository.

Keyword	Function
property:name	Finds components that define a specific property. Use *property*:color to find components that define a property called color.
event:name	Finds components that define specific event sets. Use *event*:propertyChange to find components that define an event set called propertyChange.
method:name	Finds components that define specific methods. Use *method*:createAnimation to find components that define a method called createAnimation.

Table 2: JavaBeans Special Functions for CORBA Interfaces

Keyword	Function
operation:name	Finds CORBA interfaces that include specific operations. Use *operation*:selectDrill to find a CORBA interface that defines an operation called selectDrill.
parameter:name	Finds CORBA interfaces that include parameters of a given name or type. Use *parameter*:long to find a CORBA interface that contains an operation that takes a long as a parameter.
exception:name	Finds CORBA interfaces that define specific exceptions. Use *exception* :InvalidName to find CORBA interfaces that define an exception called InvalidName.
attribute:name	Finds CORBA interfaces that define specific attributes. Use *attribute*:color to find CORBA interfaces that define an attribute called color.

Table 3: CORBA Interface-Specific Special Functions

2.4 Industry Domain

Application-specific lexicons are being developed that can be used to facilitate searches by application domains such as manufacturing, medical, and finance. In theory, these lexicons will help identify components in these domains and simplify the process of identifying suitable components. Lexicon terms may be distilled from existing component interfaces that are representative of a given domain. However, it is insufficient to parse these interfaces in a similar manner to the Agora indexer, as selected terms should be indigenous to a given domain and are not easily found outside it.

During the search process, specifying a domain causes the lexicon of domain-specific terms to be attached to the query. This allows the search engine to perform a relevancy ranking on components that match the query terms. Query terms are best given by using the "+" operator when searching a specific domain, as this ensures that these terms are found in components included in the result set.

Lexicon terms are currently selected manually in Agora due to the intelligence required to identify domain-unique terms.

3 Implementation

Agora is designed to be extensible to different component technologies, provide good performance to searchers accessing the Web site, and provide advanced searching capabilities. The budget for completing work on Agora and publishing the results is under $70,000. Resultantly, the implementation strategy that we adopted required the use of existing components as leverage. For example, the basic functionality of the AltaVista Internet service was incorporated into Agora. This service is used to identify Web pages containing Java applets. This basic functionality was extended to identify and introspect JavaBeans. Introspected interfaces are then used to build a searchable index of terms.

In addition to the AltaVista Internet Service, Agora incorporates the AltaVista Search Developer's Kit (SDK). This allows Agora to provide advanced search capabilities at considerably lower cost than custom development of these features.

The overall Agora architecture is shown in Figure 2. Independent agents are used for each component class, making the design extensible to other components models.

An agent in Agora is simply an independent process that understands a specific domain and component class. Two agents have been developed so far: a JavaBeans agent that harvests URLs from the AltaVista Internet service and a CORBA interface agent. The JavaBeans agent searches for hypertext markup language (HTML) pages containing applet tags using the AltaVista Internet service, loads and introspects these applets, and indexes the interfaces of any JavaBeans that are discovered. The CORBA interface agent uses the CORBA naming service to find CORBA interfaces and the implementation repository to discover their interfaces. A third ActiveX agent is under consideration.

The query client is implemented using Java Server Pages (JSP) on Sun's Java Web server. The use of Java Server Pages allows Java code to execute on the server to generate HTML pages that are then downloaded to the client.

Java Server pages can be extended using JavaBeans to which both explicit and implicit calls are made. Implicit calls are made to set properties within the JavaBean corresponding to input fields in the HTML form. For example, the main text input field for entering the search criteria in the Agora query interface is named `criteria`. When a POST method occurs, either because the enter key was pressed or because an input field of type submit was selected, the Java Web Server calls the `setCriteria()` member function on the imported JavaBean.

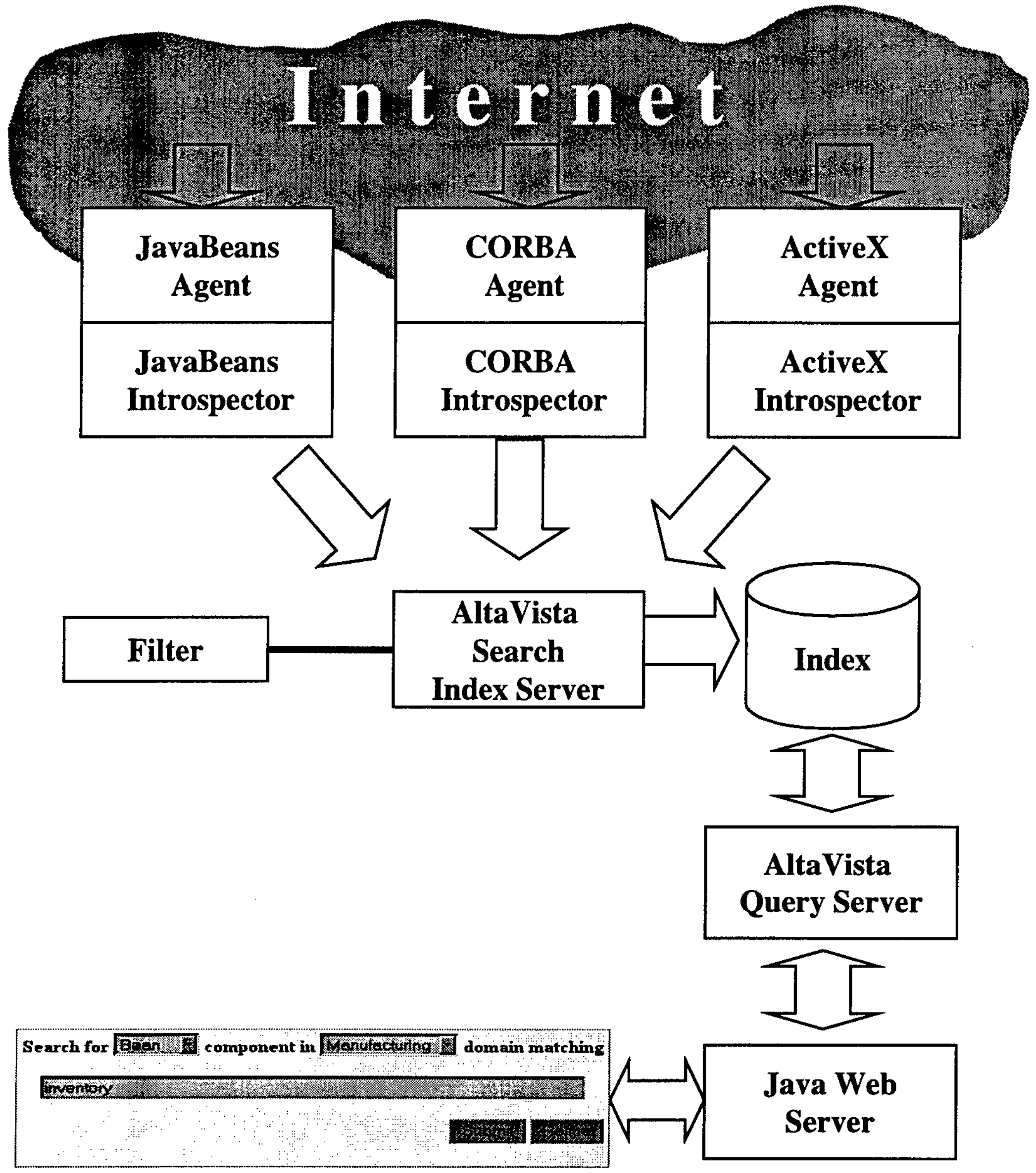


Figure 2: Agora Architecture

The existing implementation of Agora has been operationalized to identify and catalog JavaBeans over the Internet. This version is also capable of indexing CORBA interfaces, although an effective CORBA search algorithm has not yet been implemented. In addition to these component types, it is possible to develop agents that search, introspect, and catalog ActiveX controls, remote method invocation (RMI) servers, and other reusable software components. In fact, multiple agents may be developed per component class to implement different search strategies (for example, a traditional spider to search Intranets for JavaBeans).

3.1 AltaVista Search Developer's Kit (SDK)

Digital's AltaVista SDK is used by Agora for indexing and retrieving component data. The AltaVista SDK provided advanced search capabilities that would have required substantial effort to develop. Although the AltaVista SDK had many advantages, it also introduced some complexity. For example, it was necessary to implement components such as the JavaBeans agent in Java to support JavaBeans introspection. The AltaVista SDK, on the other hand, is composed of C language interfaces and libraries which are only available on Windows NT and DEC Alpha platforms. Thus we were required to implement some form of wrapper for the AltaVista SDK to make its services available to the JavaBeans agent.

Figure 3 is taken from the AltaVista SDK documentation and shows the low-level architecture of an SDK application. In Agora, the AltaVista SDK was incorporated using CORBA. A C++ wrapper was developed around the AltaVista SDK for both the index and query services. Each of these runs as a separate CORBA service in the role of the "User Application" as shown in Figure 3.

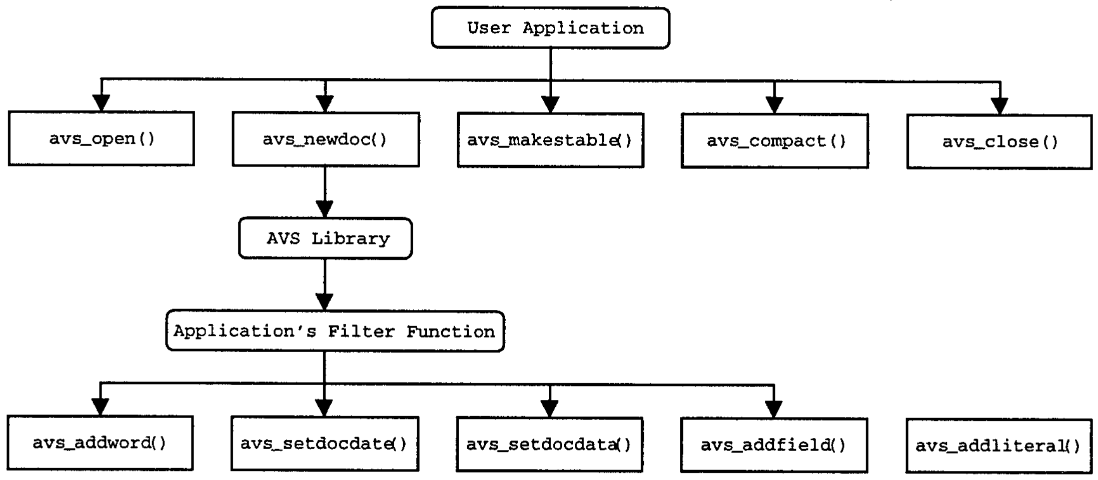


Figure 3: AltaVista SDK Architecture

Components are discovered and introspected by the agents. Interfaces are tokenized based on white space and internal capitalization. The resulting terms are concatenated in a component description string along with some meta-information, an example of which is shown below:

```
http://www.orl.co.uk/~bjm/java//SlideShow <name> SlideShow
<property> color <event> pageTurn <method> load Slide main
show Details run btn About show Captions load Next Slide
```

The component description string is passed to the index server. The index server calls the `avs_newdoc()` function, passing the component description string. The Agora-supplied filter then uses the `avs_addword()` function to add the terms contained in this string to the index, as well as the other AltaVista application program interface (API) calls shown in Figure 3 to record various properties about the document.

The component description string also contains meta-data in the form of special tags in the string delimited by '<' and '>' characters. The terms inside these delimiters correspond to special fields that can later be queried by a user of the system using the special functions provided.

The creation of this string is imposed by the architecture of the AltaVista SDK. Each component is modeled in the AltaVista index as a separate document. Since the `avs_newdoc()` function is called per document, and it, in turn, calls the filter function to add the words and fields, it is necessary to embed the meta-data in the component description string, which is then interpreted by the filter function.

3.2 JavaBeans Agent

For the implementation of the Internet JavaBeans agent, we decided to implement a meta-search engine to harvest candidate URLs from the AltaVista Internet service. The AltaVista service was selected because of its special functions for Web searches. In particular, searches of the format: "`applet:class`" can locate HTML pages containing applet tags where the code parameter is equal to specified Java applet class. For example, a search for "`applet:sine`" can be used to find applets where the code parameter is specified as "`sine`" or "`sine.class`".

The basic algorithm that we used to harvest is to query the AltaVista search engine to return pages containing applets using "applet:" as the query string. Currently, there are a total of 1,530,275 documents matching this request. AltaVista responds by generating an HTML page containing the first 10 hits from the resulting set. After parsing the page to extract the URL for these Web pages, the string "`&navig`" is appended to the end of the command sent to AltaVista along with the number of hits. This command is then sent to the AltaVista Internet service, which then returns the next 10 hits.

One problem with this approach is that the AltaVista service, either by design or by flaw, does not provide matching documents beyond the first thousand for a given query. This required that a mechanism be developed to distribute the estimated 1.5 million documents across multiple queries so that each query (or most of the queries) returns less than 1000 documents.

One approach for distributing these documents across multiple queries is to segment the documents according to the first three characters in the name of each applet. The AltaVista service supports the use of wildcards such as an asterisk (*) to broaden a search. The wildcard, however, must be proceeded by at least three characters. The number of combinations of N alphanumeric characters taken M ways can be calculated using the following formula:

$$C_{NM} = \frac{N!}{M!(N-M)!}$$

AltaVista defines a word as any string of letters and digits that is separated by either white space (such as spaces, tabs, line ends, or the start or end of a document) or special characters and punctuation (such as %, $, /, #, and _). Therefore, any given name can consist of any combination of alphanumeric characters of which there are 36 (26 alphabetic and 10 numeric).[1] These 36 characters can be combined in 7,140 unique alphanumeric strings. If distributed evenly across the 1.5 million documents, this would result in approximately 210 documents per query. Actual applet names are, of course, not evenly distributed across these groupings.

A second approach for distributing documents across multiple queries is to use the date field. Advanced searches in AltaVista can be restricted to find documents last modified during a specific date range. The JavaBeans agent uses this feature to segment the total number of documents into groups of less than 1000. Starting from a date (initially the current one), we query the AltaVista database to find the exact number of matches over the previous 30 days. The advanced search capability of the AltaVista Internet service provides a function for this that can be invoked by passing `&fmt=n` to AltaVista's `query` common gateway interface (CGI) executable file. The exact number of matches from the generated HTTP page is examined to see if it is between 0 and 1000. If the number of matching documents is greater than 1000, we reduce the date range by half and resubmit the query. Alternatively, if we find 0 matches, we expand the date range.

Once a date range that contains between 0 and 1000 hits has been identified, we harvest these documents by reissuing the search query without the `&fmt=n` parameter, bringing up the actual search results. This technique has been generally successful in finding queries that contain from 1 to 1000 entries. However, there have been cases where a single day contains more than 1000 entries. Since a single day is the limit of granularity of these queries, we must harvest the first 1000 entries and discard the remainder.

Another defect in the AltaVista Internet service is that the "exact" number of matches returned by the `&fmt=n` parameter is not always accurate. This is primarily a concern when the number returned is greater than the actual number of matching documents, since the JavaBeans agent will continue to loop in an attempt to harvest these results. This made it necessary to add a timeout mechanism so that the loop would exit after a fixed number of attempts to retrieve these matching documents from AltaVista.

[1] This does not consider internationalized use of the International Organization for Standardization (ISO) Latin-1 character set such as a word containing an accent or other diacritical mark.

The agent constructs a URL for the JavaBeans class. For example, URLs for JavaBeans consist of the Web URL, the code base, the class, and/or the archive name. This URL is maintained as the document data in the index, along with the introspected interface data.

3.3 Performance of the JavaBeans Agent

Considerable effort was spent optimizing the speed at which components could be collected. Initially, we found that the JavaBeans agent was discovering JavaBeans at low rate, and even this performance would drop off quickly.

The JavaBeans agent is implemented using the beta2 version of Java Development Kit (JDK) 1.2. This version of the JDK contains classes, such as the `URLClassLoader`, that are not available in JDK 1.1. Initially, the agent was implemented using a single thread of control. We discovered that this agent might run well for several minutes, but then it would often bog down in calls to the `URLClassLoader`. A typical run might retrieve in the vicinity of 20 JavaBeans before grinding to a halt.

To address this problem, we decided to introduce multiple threads for examining the URLs harvested from AltaVista, loading the classes with the `URLClassLoader` and performing the introspection on the resultant class. We developed a simple synchronized queue. The main program would harvest URLs from the AltaVista site and add them to the queue. Approximately 20 independent threads would then retrieve these URLs from the queue for introspection.[2]

Since loading and introspecting classes is the most central processing unit (CPU)-intensive portion of the overall component collection process, we further decided to separate this functionality out into a collection of identical RMI servers. This arrangement allows us to provide load balancing by distributing the work of loading and introspecting classes across multiple processors. Each of these remote loaders, in turn, connected to the index CORBA server running on the NT platform. The resultant architecture is shown in Figure 4. Although the index server might appear to be the bottleneck in the system, it was, in actuality, never severely taxed and no component data were lost due to time-outs from the server.

This more sophisticated arrangement helped, but we found that after retrieving an average of 40-60 JavaBeans the JavaBeans agent again began to bog down in calls to the `URLClassLoader`. We tried using the Sun supplied `sunwjit` compiler at this time, which provided an order of magnitude better runtime performance of the JavaBeans agent. However, this improved performance only served to decrease the time required to collect the components and did not affect the total number of JavaBeans retrieved.

[2] An excessive number of threads can cause the depletion of limited system resources, such as the number of available file descriptors.

At this point we looked at a number of options for achieving continual operation of the JavaBeans agent and optimizing the number of components indexed. One option was to solve the problem of the `URLClassLoader` hanging by specifying a time-out. However, the `URLClassLoader` available with the beta2 JDK 1.2 release does not provide a mechanism for specifying a time-out or a means to access the underlying socket layer. The second option was to provide some level of internal thread management to kill and restart threads that were hung.

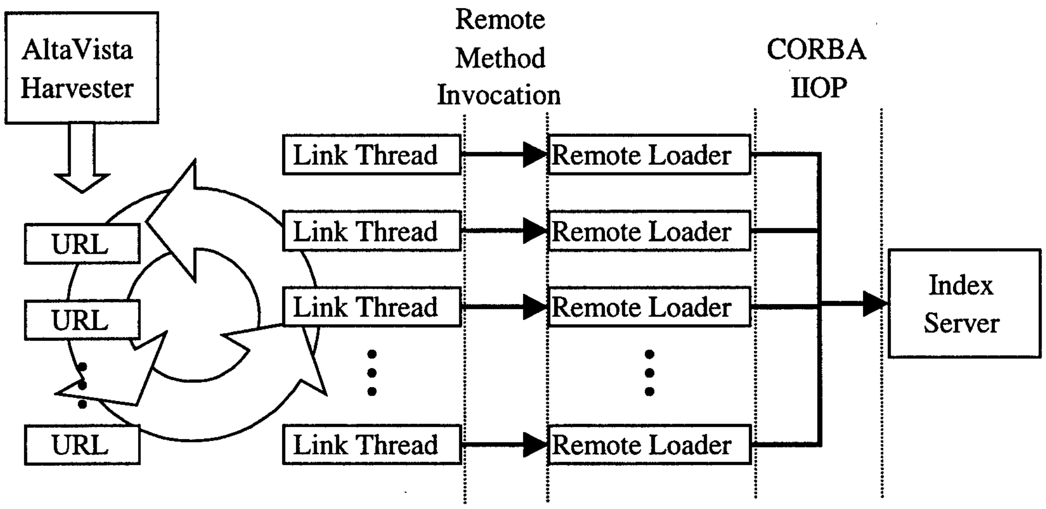


Figure 4: JavaBeans Agent

We finally decided to implement a crude but effective solution. The JavaBeans agent was modified to check point its work by recording its progress through the AltaVista database: the JavaBeans agent recorded the current date range being examined and the number of hits in that data range processed so far. The JavaBeans agent is restarted each hour (using `cron` under UNIX or the service Schedule under Windows NT) using the preserved state data. A similar approach was used in IBM's jCentral [Aviram 98] and appears to be an effective means of improving performance. In Agora we were able to retrieve over 800 JavaBeans in a 24-hour period, and an additional 1400 JavaBeans over a 48-hour period.

We expect these numbers to improve as we address other deficiencies in the implementation and integrate newer, more robust versions of the JDK.

3.4 CORBA Agent

In addition to the JavaBeans agent, a CORBA agent was implemented, although this agent is more experimental due to the lack of a CORBA infrastructure necessary to fully support the location of CORBA-based components.

3.4.1 CORBA Background

To understand the implementation of the CORBA agent, it is necessary to understand some details about CORBA itself. In CORBA, servers are implemented as distributed objects. Each of these objects is represented by an object reference. Object interfaces are defined using the Object Management Group Interface Definition Language (OMG IDL). This language is used to define the operations that may be performed on objects and the parameters to those operations. IDL is compiled into stubs that can be statically linked at compile time. It may also be translated into an interface repository that can be dynamically accessed at runtime.

CORBA servers can implement any number of interfaces and any number of objects can be created that implement a given interface. The object request broker (ORB) maintains objects in a vendor-specific manner in an implementation repository. Objects may be located in an implementation-independent fashion using the naming service.

The naming service is defined by the OMG as part of the Common Object Services or CORBA Services. The naming service allows one or more logical names to be associated with an object reference. A server that holds an object reference can register it with the naming service, giving it a name that can be subsequently used by clients of that server to find the object. Client applications can use the naming service to obtain the object reference using the logical name assigned to that object.

3.4.2 Application Bridge

The naming service provides a standard mechanism for locating CORBA objects, allowing Agora to dynamically locate objects at runtime. This is a necessary but insufficient mechanism since Agora must also be able to dynamically determine component interfaces at runtime. This problem is partially addressed by the CORBA interface repository. An interface repository contains descriptions of CORBA object interfaces. The data in an interface repository are the same as in IDL files, but the information is organized for runtime access by clients. A client can browse an interface repository or look up the interface for any object for which it has a reference.

Figure 5 illustrates the architecture of Agora's CORBA agent. The CORBA agent binds to a CORBA naming service and iterates through the values using the `BindingIterator` naming service interface. The binding name of each object is then resolved to a CORBA object. The CORBA object can be used to access interface information directly using the `get_interface()` call or indirectly using the repository identifier.

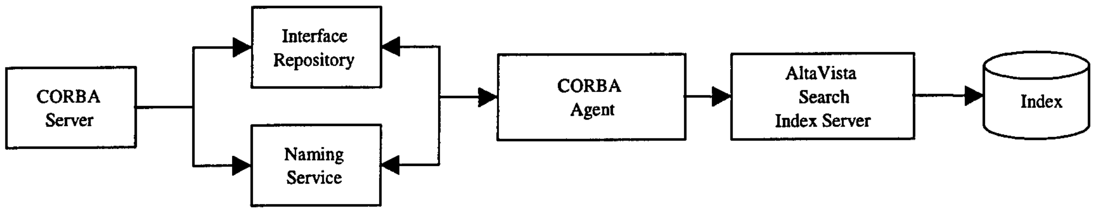


Figure 5: CORBA Agent

For each CORBA interface discovered in this fashion, we extract attribute and operation names (including associated return value, parameter names, and defined exceptions) and index the terms. As discussed previously, CORBA interfaces can be searched for using special functions to find components that define a particular operation, attribute, parameter, or exception.

The AltaVista index server was developed using version 2.3 of Iona's Orbix. The JavaBeans agent, for example, establishes a connection with the index server using the `bind()` function.

The CORBA agent was implemented to work with the Visigenics' VisiBroker implementation repository and Naming Service. VisiBroker was selected in this case due to some vendor unique services, such as the location service, that we wished to investigate.

The use of VisiBroker introduced a design problem in that the CORBA agent could not be both a VisiBroker and an Orbix client at the same time, as these libraries would clash. It is possible, but difficult, to get an Orbix client to communicate with a VisiBroker naming service, since a call to `resolve_initial_references()` to resolve the "`NameService`" can only be used to connect to the naming service supplied by the same vendor. This requires that the interoperable object reference (IOR) for the naming service be obtained independently and converted in an object reference and narrowed to the naming service.

Rather than deal with these interoperability problems, we used an application bridge to connect the index server with the CORBA agent by introducing an additional RMI server between the two processes as shown in Figure 6.

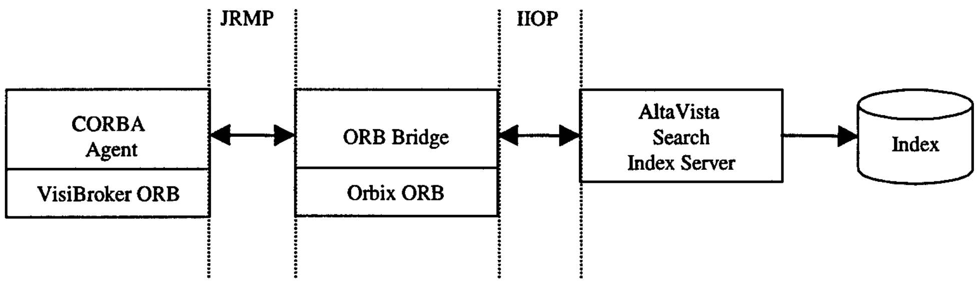


Figure 6: ORB Bridge

Like the JavaBeans agent, the ORB bridge is implemented as an Orbix client; however, it also functions as an RMI server. The CORBA agent can now be freely implemented as a VisiBroker client and communicate with the ORB Bridge via RMI calls over the Java Remote Method Protocol (JRMP). This solution was simple to implement and does not introduce any significant runtime overhead.

4 Comparison of Agora to Related Technologies

Agora can be compared and contrasted with two different technologies: Web search technology and software repository technology. There are also some interesting comparisons that can be made between developing agents for JavaBeans and CORBA interfaces.

4.1 Agora and Existing Search Engines

Principal, existing search engines provide convenient support for different kinds of Web content. Different search capabilities are provided for different types of content. For example, text content can be searched by simple but effective pattern matching, while images can be searched only by image name.

Leaders in the Internet development community are voicing concerns about the growing ineffectiveness of monolithic search engines used for a specific purpose [Aviram 98]. As the Internet grows, information associated with any given keyword grows accordingly, causing general-purpose search engines to become clogged with massive amounts of often irrelevant data.

The Agora search engine enhances existing but rudimentary search capabilities for Java applets. By using Java introspection, the Agora search engine can maintain a more structured and descriptive index that is targeted to the type of content (the component model) and the intended audience (application developers) than is supported by existing search engines. For example, information about component properties, events, and methods can be retrieved from Agora.

Developers of search engines such as AltaVista might decide to incorporate this kind of search capability; this would be a welcome indication of Agora's success. However, it is also possible that capabilities such as the Agora search engine will occupy a value-added market niche in the overall World Wide Web. For example, capabilities such as domain-specific searches may be too narrow for broad-based search engines to support profitably.

4.2 Agora and Software Repositories

A traditional approach has been to develop large-scale software repositories as large central databases containing information about components and, often, the components themselves. Examples of such systems include the Center for Computer Systems Engineering's Defense

System Repository, the JavaBeans Directory, and the Gamelan Java directory. Such efforts have historically failed, principally as a result of their conception as *centralized systems*. The reasons for these failures include limited accessibility and scalability of the repository, exclusive control over cataloged components, oppressive bureaucracy, and poor economy of scale (few users, low per-user benefits, and high cost of repository mechanisms and operations).

Agora replaces old-fashioned and costly software repositories. Agora automatically compiles indexes by going out over the Internet and discovering and collecting information about software components. Agora does this in a nonjudgmental manner, so the problem of having a sole arbiter decide what does and does not belong in the repository is eliminated. Quality assurance in the Agora model is not guaranteed—we believe that component databases need to be, at first, free and inclusive. Value-added industries such as consumer reports and underwriter labs can add value by providing independent quality assurance of popular components. This will help ensure that candidate components are identified and not simply eliminated based on the criteria of the company maintaining the repository.

5 Summary and Conclusions

Agora is designed to make it easier for system integrators to discover components that meet the requirements of their systems and provide a means for component producers to advertise their products. With Agora, components can be quickly located and evaluated as candidates for integration, eliminating an inhibitor for component-based software development. Although more work must be done to support a true commerce in software components, Agora represents a useful integration of Web search engines and component introspection.

In general, we had considerably more success locating and introspecting JavaBeans than CORBA interfaces. Locating CORBA services turned out to be problematic for several reasons. First, the majority of CORBA servers do not store their object references in a naming service. Second, even if they did, there is no good bootstrapping process for finding an initial object reference for the naming service. This problem could be addressed by having naming services respond to queries on a well-known standard port number or providing some sort of meta-naming service. The best opportunity to discover a naming service is to look for them at vendor-supplied default port numbers. Third, unlike Java applets, CORBA services are not integrated into Web pages directly, but through intermediate languages such as Java. This makes it difficult to use existing search services such as AltaVista to discover CORBA services on the Internet.

Once a CORBA server is located, it is also difficult to extract interface information since this information is not inherently part of the component, as in JavaBeans. Instead, CORBA relies on an external interface repository. Use of the interface repository is optional, and the majority of CORBA servers do not use it. Although it is apparent that CORBA servers have some understanding of operations (without resorting to the interface repository) to support Dynamic Invocation, there is no interface that provides access to this information.

Further, the necessity of establishing communication with a naming service, interface repository, and object decreases the likelihood of finding and introspecting a CORBA interface.

Agora demonstrates that it is feasible to create an automated tool for locating, introspecting, and indexing JavaBeans on the Internet. Although not yet implemented, we hypothesize that an ActiveX agent is equally plausible. For a CORBA Agora agent to be successful, the Object Management Group (OMG) must adopt a component model comparable to JavaBeans, an integrated interface repository, and a means of locating the naming service by means of a well-known port or other mechanism.

References

[Aviram 98]	Aviram, Mariva H. "Code-Centric Search Tool Strives to Reduce Java Development Time." *JavaWorld* [online]. Available WWW <URL: http://www.javaworld.com/jw-06-1998/jw-06-jcentral.html> (June 1998).
[Hurwitz 98]	Hurwitz, Shirley M. *Interoperable Infrastructures for Distributed Electronic Commerce*. Available WWW <URL: http://www.atp.nist.gov/atp/98wp-ecc.htm>.
[Maudlin 97]	Maudlin, Michael. "Lycos: Design Choices in an Internet Search Service." *IEEE Expert* 12, 1 (January-February 1997).
[NIST 98]	National Institute of Standards and Technology. *Framework for Discrete Parts Manufacturing*. Available WWW <URL: http://www.mel.nist.gov/namt/projects/framwork/fw1.htm>.
[Nowak 97]	Nowak, Michael. *ATP Focused Program: Component-Based Software*. Available WWW <URL: http://www.atp.nist.gov/atp/focus/cbs.htm> (November 1997).
[SIMA 97]	*Systems Integration for Manufacturing Applications*. Available WWW <URL: http://www.mel.nist.gov/msid/sima/sima.htm>.
[Webster 96]	Webster, Kathleen & Paul, Kathryn. "Beyond Surfing: Tools and Techniques for Searching the Web." *Information Technology* (January 1996).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE August 1998	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Agora: A Search Engine for Software Components		5. FUNDING NUMBERS C — F19628-95-C-0003
6. AUTHOR(S) Robert C. Seacord, Scott A. Hissam, Kurt C. Wallnau		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-98-TR-011
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/DIB 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-98-011
11. SUPPLEMENTARY NOTES		
12.A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Agora is a software prototype being developed by the Commercial Off-the-Shelf (COTS)-Based Systems Initiative at the Software Engineering Institute (SEI). The object of this work is to create an automatically generated and indexed worldwide database of software products classified by component model. Agora combines introspection with Web search engines to reduce the costs of bringing software components to, and finding components in, the software marketplace. This report describes Agora's role in an emerging component industry and the features and capabilities provided by Agora. The implementations of a JavaBeans agent and a Common Object Request Broker Architecture (CORBA) agent are also described. These agents are used to gather components of their respective types.

14. SUBJECT TERMS Agora, Common Object Request Broker Architecture (CORBA), database, commercial off-the-shelf components, introspection, JavaBeans, search engines, World Wide Web (WWW)	15. NUMBER OF PAGES 30
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102